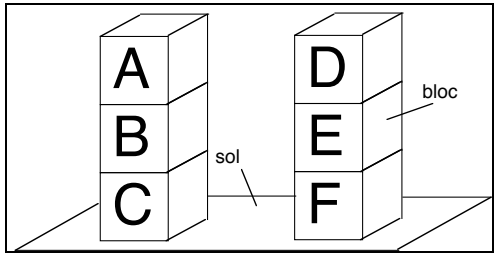
Exemple de programmation en CLIPS :

le monde des blocs

Bernard ESPINASSE
Professeur à l'Université d'Aix-Marseille
2008

Exemple de programmation en CLIPS : le monde des blocs

- un bon exemple de plannification pouvant être appliqué à la robotique
- il s'agit de bouger des blocs pour obtenir une configuration souhaitée
- 2 piles de 6 blocs :



un exemple de but : mettre le bloc C (?upper) au dessus du bloc E (?lower) ?

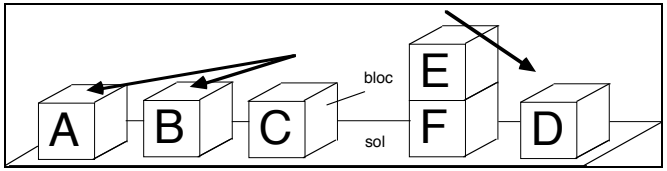
- on ne peut bouger qu'un bloc à la fois
- pour bouger un bloc il faut qu'il soit au sommet de la pile
- un bloc au sommet peut être mis au sommet de l'autre pile ou mis au sol

Mettre le bloc C (?upper) au dessus du bloc E (?lower) ?

- possible directement si C et E n'ont pas de bloc(s) au dessus d'eux par la règle en pseudocode :

```
REGLE MOVE-DIRECTLY
SI   le but est de mettre un bloc ?upper au dessus d'un bloc ?lower et
     que le bloc ?upper est au dessus de sa pile et
     que le bloc ?upper est au dessus de sa pile et
ALORS
     mettre bloc ?upper sur le dessus du bloc ?lower
```

- *mais ici pas possible : pour appliquer la règle R1, il faut que les blocs A, B et D soient dépilés sur le sol ! :*



Mettre le bloc C (?upper) au dessus du bloc E (?lower) ?

Pour que les blocs A, B et D soient dépilés sur le sol, on utilise les règles :

1 - règle pour enlever les blocs A et B (?above) au dessus du bloc à bouger C (?upper):

```
REGLE CLEAR-UPPER-BLOC
SI   le but est de mettre un bloc ?X et
     que bloc ?X n'est pas le bloc au sommet de la pile et
     que bloc ?above est au dessus du bloc ?X
ALORS
     le but est de mettre le bloc ?above sur le sol
```

=> sous but : mettre A et B au sol

2 - règle pour enlever les blocs D (?above) au dessus du bloc à recouvrir E (?lower) :

```
REGLE CLEAR-LOWER-BLOC
SI   le but est de mettre un autre bloc au dessus du bloc ?X et
     que bloc ?X n'est pas le bloc du sommet de la pile et
     que bloc ?above est au dessus du bloc ?X
ALORS
     le but est de mettre le bloc ?above sur le sol
```

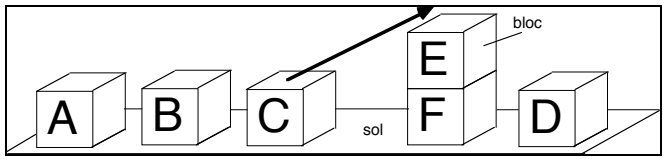
=> sous but : mettre D au sol

Mettre le bloc C (?upper) au dessus du bloc E (?lower) ?

3 • ces sous-buts étant définis, pour les atteindre, cad mettre A, B et D au sol, on utilise la règle :

```
REGLE MOVE-TO-FLOOR
SI   le but est de mettre le bloc ?upper sur le sol et
     que bloc ?upper est au sommet de sa pile
ALORS
     mettre le bloc ?upper sur le sol
```

=> A, B et C au sol, on peut alors par la première règle mettre D sur E.



Les faits en Clips nécessaires

- lorsque les règles sont écrites en speudocode, il faut déterminer (pas toujours facile) **les faits en Clips qui seront utilisés par ces règles**
- définition des blocs :

```
(bloc A)
(bloc B)
(bloc C)
(bloc D)
(bloc E)
(bloc F)
```

- sur quels blocs sont au dessus d'autres blocs est crucial (type template):

```
(deftemplate on-top-of
 (slot upper)
 (slot lower))
```

- les faits décrits par ce template doivent être :

```
(on-top-of (upper A) (lower B))
(on-top-of (upper B) (lower C))
(on-top-of (upper D) (lower E))
(on-top-of (upper E) (lower F))
```

- de même que les faits (nothing et floor n'ont pas de signification particulière):

```
(on-top-of (upper nothing) (lower A))
(on-top-of (upper C) (lower floor))
(on-top-of (upper nothing) (lower D))
(on-top-of (upper F) (lower floor))
```


Les faits en Clips nécessaires

soit au total en Clips :

```
(deffacts    initial-state
(block  A)
(block  B)
(block  C)
(block  D)
(block  E)
(bl'ock F)
(on-top-of   (upper nothing) (lower A))
(on-top-of   (upper A) (lower B))
(on-top-of   (upper B) (lower C))
(on-top-of   (upper C) (lower floor))
(on-top-of   (upper nothing) (lower D))
(on-top-of   (upper D) (lower E))
(on-top-of   (upper E) (lower F))
(on-top-of   (upper F) (lower floor))
(goal (move C) (on-top-of E)))
```


Les règles en Clips nécessaires (1)

```
REGLE MOVE-DIRECTLY
  SI le but est de mettre un bloc ?upper au dessus d'un bloc ?lower et
     que le bloc ?upper est au dessus de sa pile et
     que le bloc ?upper est au dessus de sa pile et
  ALORS
     mettre bloc ?upper sur le dessus du bloc ?lower
```

- **traduction en Clips :**

```
(defrule move-directly
 ?goal <- (goal (move ?block1)
   (on-top-of ?block2))
 (block ?block1)
 (block ?block2)
 (on-top-of (upper nothing) (lower ?block1))
 ?stack-1 <- (on-top-of  (upper ?block1)
             (lower ?block3))
 ?stack-2 <- (on-top-of (upper nothing)
        (lower ?block2))
 =>
 (retract ?goal ?stack-1 ?stack-2)
 (assert (on-top-of  (upper ?block1)
        (lower ?block2))
 (on-top-of (upper nothing)
        (lower ?block3)))
 (printout t ?block1 Il moved on top of ?block2 "." crlf))
```

Les règles en Clips nécessaires (2)

REGLE CLEAR-UPPER-BLOC

SI le but est de mettre un bloc ?X et
que bloc ?X n'est pas le bloc au sommet de la pile et
que bloc ?above est au dessus du bloc ?X
ALORS
le but est de mettre le bloc ?above sur le sol

• **traduction en Clips :**

```
(defrule clear-upper-block
 (goal (move ?block1))
 (block ?block1)
 (on-top-of (upper ?block2) (lower ?block1))
 (block ?block2)
 =>
 (assert (goal  (move ?block2)
    (on-top-of floor))))
```


Les règles en Clips nécessaires (3)

REGLE CLEAR-LOWER-BLOC

SI le but est de mettre un autre bloc au dessus du bloc ?X et
que bloc ?X n'est pas le bloc du sommet de la pile et
que bloc ?above est au dessus du bloc ?X
ALORS
le but est de mettre le bloc ?above sur le sol

• **traduction en Clips :**

```
(defrule clear-lower-block
 (goal (on-top-of ?block1))
 (block ?block1)
 (on-top-of (upper ?block2) (lower ?block1))
 (block ?block2)
 =>
 (assert (goal  (move ?block2)
    (on-top-of floor))))
```


Les règles en Clips nécessaires (4)

REGLE MOVE-TO-FLOOR

SI le but est de mettre le bloc ?upper sur le sol et
que bloc ?upper est au sommet de sa pile
ALORS
mettre le bloc ?upper sur le sol

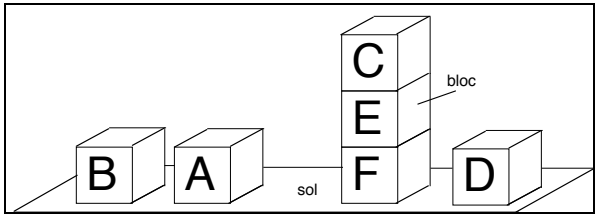
• **traduction en Clips :**

```
(defrule move-to-floor
 ?goal <- (goal (move ?block1) (on-top-of floor))
 (block ?block1)
 (on-top-of (upper nothing) (lower ?block1))
 ?stack <- (on-top-of   (upper ?block1)
        (lower ?block2))
 =>
 (retract ?goal ?stack)
 (assert    (on-top-of  (upper ?block1)
        (lower floor))
  (on-top-of    (upper nothing)
        (lower ?block2)))
 (printout t ?block1 " moved on top of floor." crlf))
```


Exécution du programme des blocs

• **entrée/sortie d'écran :**

```
CLIPS> (unwatch all) ↲
CLIPS> (reset) ↲
CLIPS> (run) ↲
A moved on top of floor.
B moved on top of floor.
D moved on top of floor.
C moved on top of E.
CLIPS>
```



```
;;;================================
;;;   Blocks World Program (1)
;;;     This program was introduced in Chapter 8.
;;;     CLIPS Version 6.0 Example
;;;     To execute, merely load, reset and run.
;;;================================

    (deffacts initial-state
     (block A)
     (block B)
     (block C)
     (block D)
     (block E)
     (bl'ock F)
     (on-top-of (upper nothing) (lower A))
     (on-top-of (upper A)    (lower B))
     (on-top-of (upper B)    (lower C))
     (on-top-of (upper C)    (lower floor))
     (on-top-of (upper nothing) (lower D))
     (on-top-of (upper D)    (lower E))
     (on-top-of (upper E)    (lower F))
     (on-top-of (upper F)    (lower floor))
     (goal (move C) (on-top-of E)))
```


Base de connaissance finale (suite)

```
(defrule move-directly
 ?goal <-   (goal   (move   ?block1)
    (on-top-of   ?block2))
 (block ?block1)
 (block ?block2)
 (on-top-of   (upper nothing) (lower     ?block1))
 ?stack-1 <-     (on-top-of   (upper   ?block1)
        (lower   ?block3))
 ?stack-2 <-     (on-top-of   (upper   nothing)
        (lower   ?block2))
 =>
 (retract ?goal ?stack-1 ?stack-2)
 (assert     (on-top-of   (upper   ?block1)
        (lower   ?block2))
  (on-top-of    (upper   nothing)
        (lower   ?block3)))
 (printout t ?block1 Il moved on top of  ?block2 "." crlf))

(defrule clear-upper-block
 (goal  (move   ?block1))
 (block     ?block1)
 (on-top-of (upper   ?block2)    (lower   ?block1))
 (block     ?block2)
 =>
 (assert    (goal   (move   ?block2)
    (on-top-of   floor))))
```


Base de connaissance finale (fin)

```
(defrule clear-lower-block
 (goal  (on-top-of ?block1))
 (block     ?block1)
 (on-top-of (upper ?block2) (lower ?block1))
 (block     ?block2)
 =>
 (assert    (goal   (move   ?block2)
    (on-top-of   floor))))

(defrule move-to-floor
 ?goal <-   (goal   (move ?block1)  (on-top-of floor))
 (block     ?block1)
 (on-top-of (upper nothing) (lower ?block1))
 ?stack <-  (on-top-of      (upper ?block1)
            (lower ?block2))
 =>
 (retract   ?goal   ?stack)
 (assert    (on-top-of   (upper   ?block1)
        (lower   floor))
  (on-top-of    (upper   nothing)
        (lower ?block2)))
 (printout t ?block1 " moved on top of floor." crlf))
```


Base de connaissance finale globale

```
;;;================================
;;;   Blocks World Program (1)
;;;     This program was introduced in Chapter 8.
;;;     CLIPS Version 6.0 Example
;;;     To execute, merely load, reset and run.
;;;================================

(deffacts initial-state
  (block A)
  (block B)
  (block C)
  (block D)
  (block E)
  (bl'ock F)
  (on-top-of  (upper nothing) (lower A))
  (on-top-of  (upper A)     (lower B))
  (on-top-of  (upper B)     (lower C))
  (on-top-of  (upper C)     (lower floor))
  (on-top-of  (upper nothing) (lower D))
  (on-top-of  (upper D)     (lower E))
  (on-top-of  (upper E)     (lower F))
  (on-top-of  (upper F)     (lower floor))
  (goal (move C) (on-top-of E)))

(defrule move-directly
  ?goal <-     (goal  (move ?block1)
          (on-top-of ?block2))
  (block ?block1)
  (block ?block2)
  (on-top-of   (upper nothing) (lower ?block1))
  ?stack-1 <- (on-top-of     (upper ?block1)
              (lower ?block3))
  ?stack-2 <- (on-top-of     (upper nothing)
              (lower ?block2))
  =>
  (retract ?goal ?stack-1 ?stack-2)
  (assert     (on-top-of     (upper ?block1)
              (lower ?block2))
    (on-top-of       (upper nothing)
              (lower ?block3)))
  (printout t ?block1 Il moved on top of ?block2 "." crlf))

(defrule clear-upper-block
  (goal       (move  ?block1))
  (block      ?block1)
  (on-top-of  (upper ?block2)      (lower ?block1))
  (block      ?block2)
  =>
  (assert     (goal  (move ?block2)
       (on-top-of floor))))

(defrule clear-lower-block
  (goal       (on-top-of ?block1))
  (block      ?block1)
  (on-top-of  (upper ?block2)      (lower ?block1))
  (block      ?block2)
  =>
  (assert     (goal  (move ?block2)
       (on-top-of floor))))

(defrule move-to-floor
  ?goal <-    (goal  (move ?block1)       (on-top-of floor))
  (block      ?block1)
  (on-top-of  (upper nothing)      (lower ?block1))
  ?stack <-   (on-top-of     (upper ?block1)
              (lower ?block2))
  =>
  (retract    ?goal  ?stack)
  (assert     (on-top-of     (upper ?block1)
              (lower floor))
    (on-top-of       (upper nothing)
              (lower ?block2)))
  (printout t ?block1 " moved on top of floor." crlf))
```

Base de connaissance finale (2) en utilisant les "multifield wildcards"

```
;;;======================================
;;;   Blocks World Program (2)
;;;     This program was introduced in Chapter 8.
;;;     CLIPS Version 6.0 Example
;;;     To execute, merely load, reset and run.
;;;======================================

(deftemplate goal (slot move) (slot on-top-of))

(deffacts initial-state
   (stack A B C)
   (stack D E F)
   (goal (move C) (on-top-of E))
   (stack))

(defrule move-directly
   ?goal <- (goal (move ?block1) (on-top-of ?block2))
   ?stack-1 <- (stack ?block1 $?rest1)
   ?stack-2 <- (stack ?block2 $?rest2)
   =>
   (retract ?goal ?stack-1 ?stack-2)
   (assert (stack $?rest1))
   (assert (stack ?block1 ?block2 $?rest2))
   (printout t ?block1 " moved on top of " ?block2 "." crlf))
```



```
(defrule move-to-floor
   ?goal <- (goal (move ?block1) (on-top-of floor))
   ?stack-1 <- (stack ?block1 $?rest)
   =>
   (retract ?goal ?stack-1)
   (assert (stack ?block1))
   (assert (stack $?rest))
   (printout t ?block1 " moved on top of floor." crlf))

(defrule clear-upper-block
   (goal (move ?block1))
   (stack ?top $? ?block1 $?)
   =>
   (assert (goal (move ?top) (on-top-of floor))))

(defrule clear-lower-block
   (goal (on-top-of ?block1))
   (stack ?top $? ?block1 $?)
   =>
   (assert (goal (move ?top) (on-top-of floor))))
```


Base de connaissance finale (2) en utilisant les "multifield wildcards"

```
;;;============================
;;;   Blocks World Program (2)
;;;     CLIPS Version 6.0 Example
;;;     To execute, merely load, reset and run.
;;;============================

(deftemplate goal (slot move) (slot on-top-
of))

(deffacts initial-state
   (stack A B C)
   (stack D E F)
   (goal (move C) (on-top-of E))
   (stack))

(defrule move-directly
     ?goal <- (goal (move ?block1) (on-top-of
?block2))
   ?stack-1 <- (stack ?block1 $?rest1)
   ?stack-2 <- (stack ?block2 $?rest2)
   =>
   (retract ?goal ?stack-1 ?stack-2)
   (assert (stack $?rest1))
   (assert (stack ?block1 ?block2 $?rest2))
   (printout t ?block1 " moved on top of "
             ?block2 "." crlf))
```

```
(defrule move-to-floor
     ?goal <- (goal (move ?block1) (on-top
floor))
   ?stack-1 <- (stack ?block1 $?rest)
   =>
   (retract ?goal ?stack-1)
   (assert (stack ?block1))
   (assert (stack $?rest))
     (printout t ?block1 " moved on top
floor." crlf))

(defrule clear-upper-block
   (goal (move ?block1))
   (stack ?top $? ?block1 $?)
   =>
      (assert (goal (move ?top) (on-top
floor))))

(defrule clear-lower-block
   (goal (on-top-of ?block1))
   (stack ?top $? ?block1 $?)
   =>
      (assert (goal (move ?top) (on-top
floor))))
```